

FACULTAD DE INGENIERÍA

ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA DE SISTEMAS

**Tema:**

Código Sucio

**Autores:**

Mamani Valencia, Jeancarlos

Merino Maceta, Carlos Brayan

Morales Carlos, Jesús

Orellana Aliano, Roberto Sebastian

**Docente:**

Coronel Castillo, Eric Gustavo

**Año:**

**Año del Buen Servicio al Ciudadano**

Lima - Perú

2017

**ÍNDICE**

RESUMEN3

ANTECEDENTES4

CODIGO SUCIO6

Nombre sin Sentido

Funciones

COMENTARIOS8

Balbucear

Comentarios sobrantes

Marcadores de posición

Código comentado

Información no local

FORMATO9

Formato vertical inadecuado

Lo contrario a la metáfora del periódico

Inadecuada apertura vertical entre conceptos

Distancia vertical errónea

Formato horizontal inadecuado

Sangrado

Las reglas equipo

OBJETOS Y ESTRUCTURAS DE DATOS10

PROCESAR ERRORES11

Características Fundamentales

SISTEMAS13

Proxies de java

AspectJ

Optimizar la toma de decisiones

Estándares

Lenguajes específicos del dominio

EMERGENCIA14

Duplicados

Expresividad

**CONCURRENCIA14**

CONCLUSIONES15

RECOMENDACIONES16

REFERENCIAS BIBLIOGRÁFICAS17

**RESUMEN**

* Como se ha mostrado, el sistema operativo de nuestra computadora, al principio el programa puede estar funcionando bien, pero cuando te pida agregar más cosas al programa esto causará errores a los usuarios ya que los códigos están regados por ahí y eso causa un mal funcionamiento.
* También se ha mostrado al no ejecutar programas de prueba se hace más sencillo que el código sucio esté presente en el software.
* Además de que existen varias formas de que esto suceda, sin embargo, la mayoría de las personas allegadas a un software les da pereza corregir y hacer un buen código, en otras palabras, no prestan atención al orden, ni a los detalles, solo importándoles que genere resultados.

**ANTECEDENTES**

* Dada a la recopilación de información brindada por cada integrante de nuestro grupo obtuvimos un trabajo factible. Este estudio de gran contenido demostró la importancia que tiene hacer un buen código.
* Esta investigación está relacionada con el curso ya que es un material de conocimientos para la ilustración de procesamiento de datos a través de la recopilación de información de todos los integrantes.
* El estudio de código sucio demuestra la importancia de cada función de cada una de estas. Dado que cada función se ha visto de importancia para el usuario, se ha visto como algo muy necesario su presentación conforme se ha avanzado esta investigación.
* Este trabajo muestra cómo se debería aplicar un buen código, pero centrándose en lo malo del código y lo imprescindible que es para la solicitud de seguimientos dentro de un sistema a través de una intensa investigación y búsqueda de información de los integrantes participantes.

**CÓDIGO SUCIO**

Muchas veces al programar perdemos la intención de hacer un buen código, solo importa que el programa bote el resultado esperado.

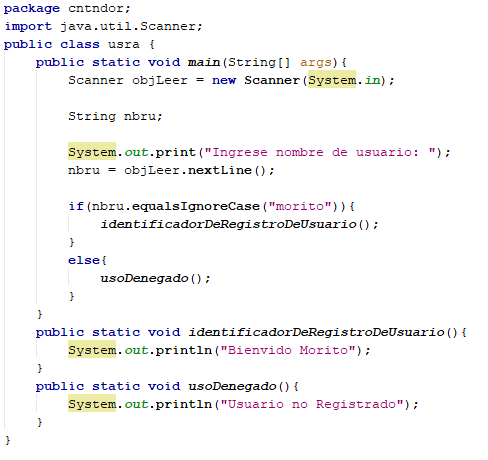
Solemos engañarnos pensando que si el programa da buenos resultados ya está terminado, para que un programa esté bien terminado debe estar bien codificado.

“Digamos que trabajas para una empresa de diseño tu jefe te solicito un programa de dibujo vectorial, al principio el programa puede estar funcionando bien, pero cuando te pida agregar más cosas al programa esto causará errores a los usuarios ya que los códigos están regados por ahí y eso causa un mal funcionamiento, al final te sacarán del proyecto y contrataran a otro, cuando el vea tus códigos no entenderá nada todo es un basurero en tu programa lo más probable es que inicien el proyecto desde cero.”

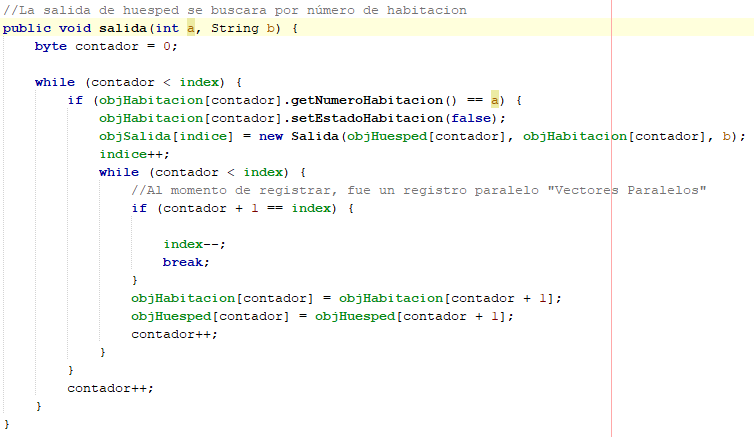
* Recuerda tu eres como un autor y el resto de personas “Programadores”, leerán tu Obra.
* En los siguientes temas a tratar veremos todos los errores que cometemos al programar.

**• Nombre sin Sentido**

* Aparecen en variables, funciones, clases y paquetes.
* La mayoría de los programadores suelen escribir sus variables con una sola letra ‘a’, mayormente lo hacen los programadores inteligentes, es como si dijeran soy inteligente yo sé que significa todas mis variables, esto trae consecuencia debido a que si otro programador ve tus códigos no entenderá nada, suele pasar que en una determinada empresa tenga su propio lenguaje de codificación puede ser que tengan una palabra clave para un variable **nbru** significa nombre de usuario, esto obliga al nuevo programador tener que aprenderse todos los códigos que use la empresa, esto causa incomodidad al nuevo empleado, sería más fácil si usaran variables que den un significado de para qué son usados. Solemos escribir nombre a las funciones que no tienen nada que ver con lo que la función que cumplen podría ser que la verifique si la cuenta está registrada o no en el sistema o ponerle un nombre muy inteligente **Identificar De Registro De Usuari**o. Lo mismo sucede con los paquetes y clases ponemos nombres que no tienen ningún sentido.



**• Funciones**

* Si tu función repite códigos tiene decenas de if, for, while, etc. El nombre de tu función es sustantivo, seguro el argumento de tu función no tienen ninguna relación con la función déjame decirte que tu “Código Apesta”. Gran parte de los programadores suelen pensar que si tu función hace más cosas es lo mejor pues déjame decirte que acabas de crear un monstruo.

**COMENTARIOS**

Como se sabe, los comentarios pueden ayudar mucho a los programadores a entender los códigos, y más aún cuando se trata de proyectos con equipos o al dar mantenimiento a los programas. Pero esto no siempre es así. En ocasiones, muchas ocasiones, tendemos a empeorar más la manera de expresar el código que no colocando comentarios. Los comentarios en sí ya muestran que hay dificultades y preocupan más a los programadores.

* No hay nada más útil que un código que está bien colocado, así como no hay nada más dañino que un comentario que muestre mentiras y desinformación.
* Los comentarios en sí demuestran nuestra incapacidad para expresarnos en el código. Los comentarios incorrectos suelen ser excusas de código pobre o justificaciones de decisiones insuficientes.

**• Balbucear.**

* Se refiere a comentarios sin razón porque el proceso lo requiere.

**• Comentarios sobrantes.**

* Comentarios colocados para describir métodos o atributos que se entienden sin necesidad de que estas estén. Y más aún cuando no se colocan para describir una línea más compleja.
* Estos comentarios son tan inservibles que tendemos a ignorarlos. Esto afecta a que otros comentarios también puedan ser ignorados al igual que los sobrantes.

**• Marcadores de posición.**

* Como se mencionó antes, agregar comentarios de más hace que se creen muchos comentarios y hace que se ignoren y así pierden su importancia y objetivo de expresar. Un claro ejemplo es comentar algo como: // Acciones //////////////////////////////

**• Código comentado.**

* Su mismo nombre ya lo dice. Escribir un código en los comentarios es de lo peor.

**• Información no local.**

* Comentarios que no describen el código que las rodea.

**FORMATO**

El formato es un código importante, tiende a ayudar a los lectores para que no se pierdan, así como también puede perderlos y darles repelús. La mala legibilidad hace que no se pueda entender de manera adecuada los códigos aún si estos son buenos códigos.

**• Formato vertical inadecuado.**

* Cuando hay demasiadas líneas de código hace engorroso buscar el método que uno busca.

**• Lo contrario a la metáfora del periódico.**

* Es que al inicio no te indique de qué se trata el código, así no dará ganas de leerlo. Que el título no sea sencillo puede hacer que el lector no sepa si está en el módulo correcto o no.

**• Inadecuada apertura vertical entre conceptos.**

* Cuando las líneas de código están todas juntas sin dejar líneas en blanco que separen a los métodos, por ejemplo. Así se oscurece su legibilidad, no estaría separando conceptos.

**• Distancia vertical errónea.**

* Quiere decir que cuando se va a una clase pasando de función en función para saber de qué se trata, termine perdiéndose debido a que las funciones relacionadas a otras están en otro lado o incluso en el otro extremo y no cerca de estas.

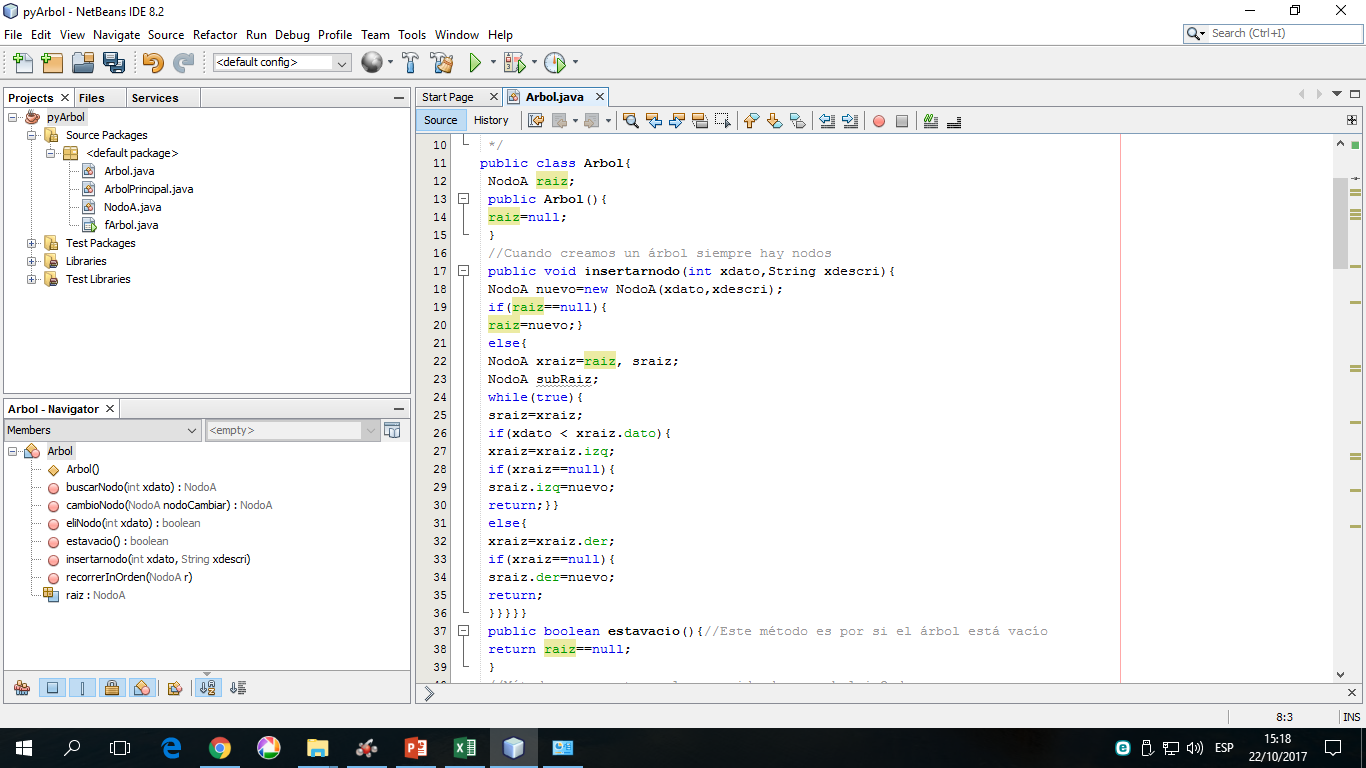
**• Formato horizontal inadecuado.**

* En muchas ocasiones uno se encuentra con líneas de código extensas que superan la cantidad de caracteres de ancho que uno puede ver. Pero pasarse de 200 caracteres ya es demasiado e inaceptable. Es un verdadero código sucio.

**• Sangrado.**

* El sangrado en el código sucio se representa por no respetar la jerarquía de un archivo de código.

**• Las reglas equipo.**

* Trabajar sin organizarse con el equipo de trabajo y hacer con tu propio formato, además que el resto del equipo lo haga a su manera es un verdadero código sucio. Con eso se está mutando en código y generará problemas con el resto del equipo.

**OBJETOS Y ESTRUCTURAS DE DATOS**

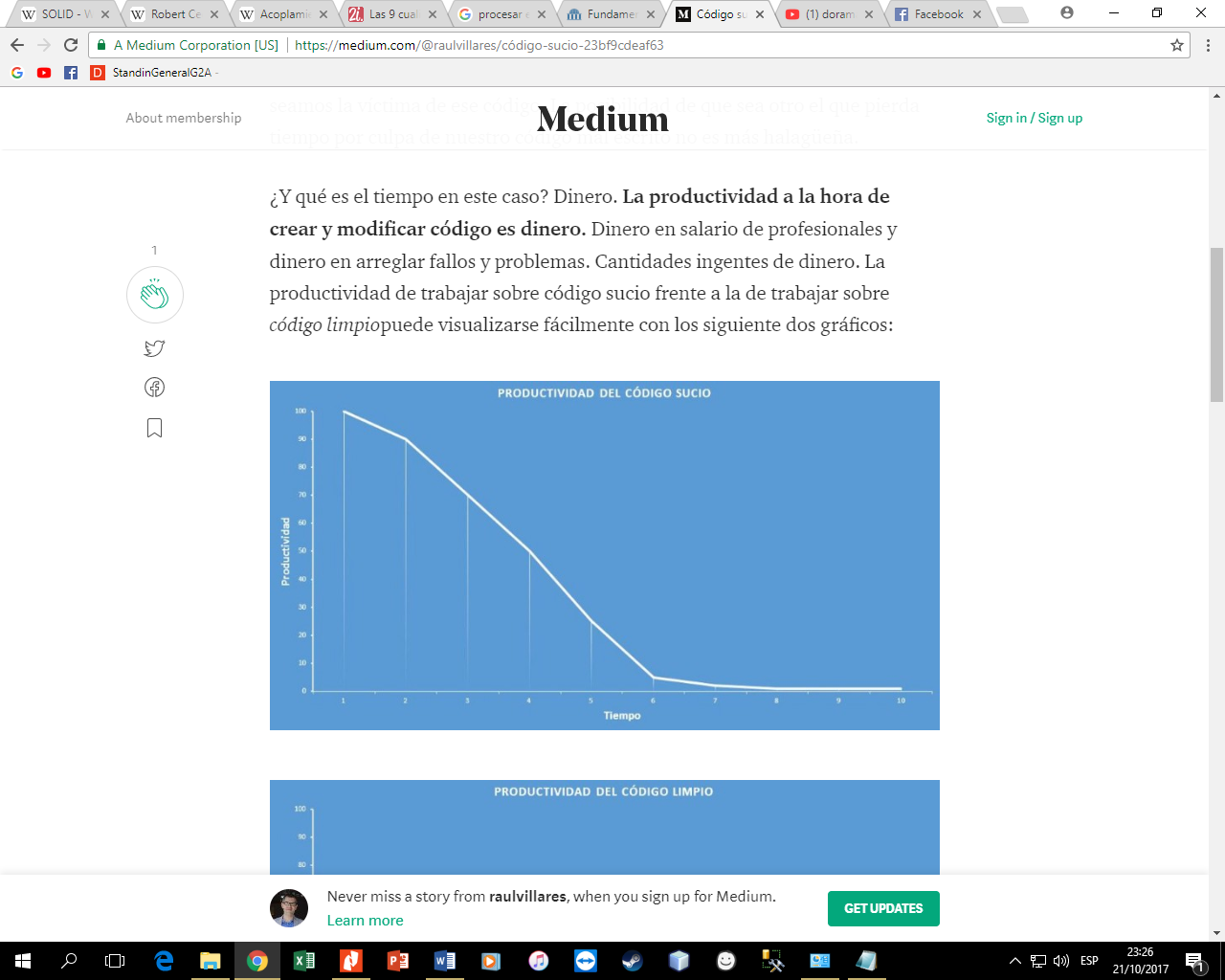
Tanto la metodología orientada a objetos como las estructuras de datos tienen sus ventajas y desventajas en algunos puntos. Hacer un código orientado a objetos para cambiar el comportamiento de objetos es inadecuado. Así como hacer un código de estructuras de datos para incluir nuevas estructuras de datos existentes también es erróneo.

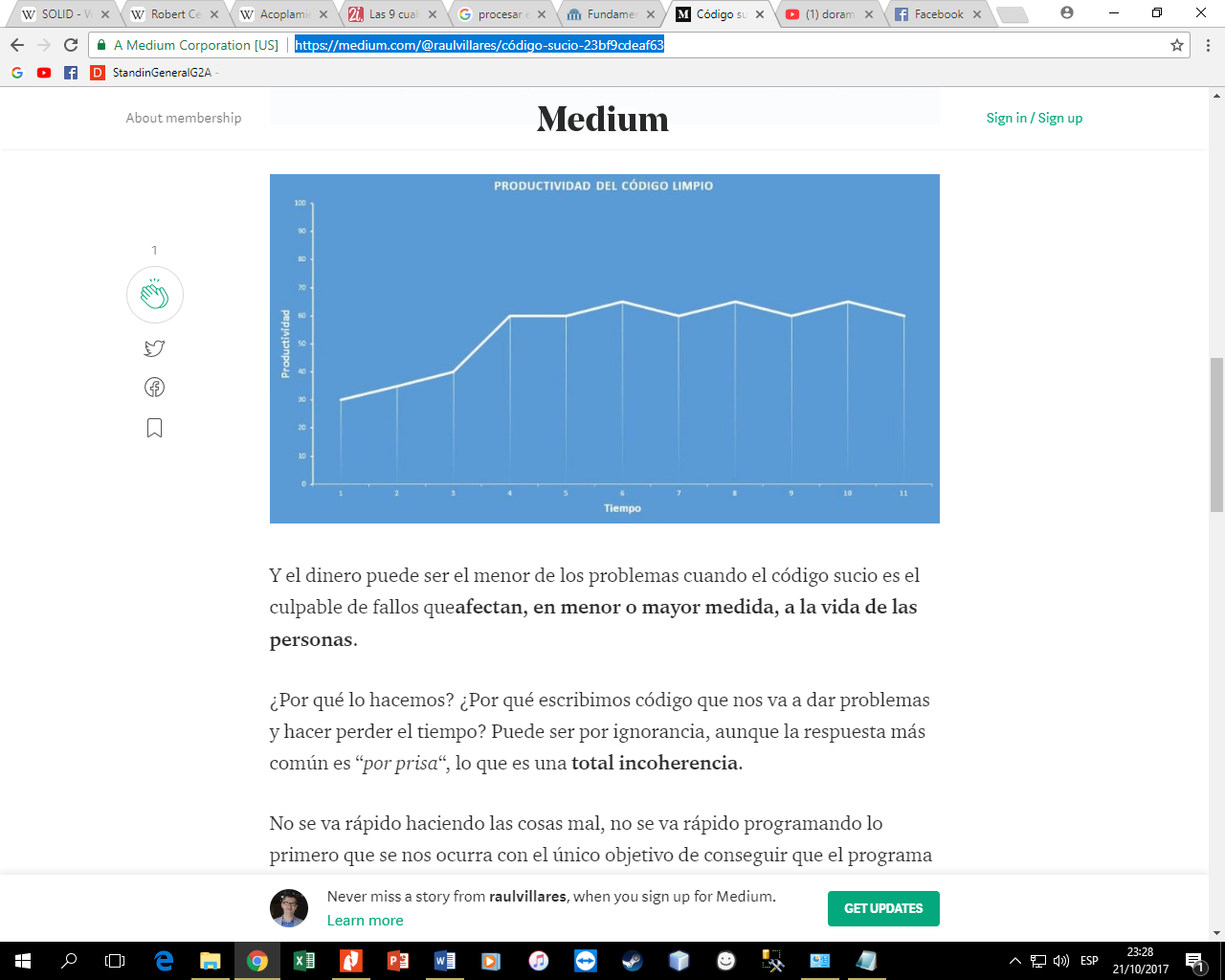
También es un código sucio usar a ambos y hacer híbridos, ya que dificulta la inclusión de nuevas funciones y también de estructuras de datos.

**PROCESAR ERRORES**

El control de errores es algo que todos debemos hacer al programar. Las entradas pueden ser incorrectas y los dispositivos pueden fallar. El programador debe comprobar inmediatamente cuando termina y/o ejecuta los procesos. De no ser así se pierde demasiado tiempo por culpa de un código mal escrito y/o por no realizar un control de calidad. La productividad a la hora de crear y modificar código es dinero.

Debemos siempre verificar nuestro código, para evitar aquellos problemas que más adelante se hacen más complejos y complicados.





**• Características Fundamentales**

* **Rigidez,** Al cambiar una pequeña parte del código, nos vemos obligados a cambiar otras muchas partes que en principio no deberían verse afectadas. La rigidez es la responsable de la típica sensación de perder mucho más tiempo.
* **Fragilidad** es la tendencia del código a romperse de manera impredecible en varios sitios al realizar un cambio.
* **No reutilizable,** A pesar de que el código podría sernos útil en un nuevo módulo o sistema, no nos atrevemos a reutilizarlo ya que la dificultad de hacerlo es demasiado grande y propensa a dar errores y problemas. Preferimos empezar de cero.

**SISTEMAS**

Son elementos únicos a comparación de los sistemas físicos. Estos deben ser limpios, una estructura invasiva afecta a la lógica de dominios y a la agilidad. Si su lógica de dominios se ve afectada la calidad se resiente, ya que los errores se ocultan y las historias son más difíciles de implementar.

**• Proxies de java**

* Son útiles en casos sencillos, como envolver invocaciones de métodos en clases concretas. son obstante, los proxies dinámicos proporcionados por el jdk solo funcionan con interfaces.

**• AspectJ**

* Es una extensión de java que ofrece compatibilidad de primer nivel para aspectos como construcciones de modularidad. Sin embargo, este ofrece un conjunto de herramientas avanzadas y complejas para la separación de aspectos. El inconveniente de AspectJ es la necesidad de adoptar nuevas herramientas y aprender nuevas construcciones del lenguaje.

**• Optimizar la toma de decisiones**

* En un sistema suficientemente amplio, ya sea una ciudad o un proyecto de software, no debe haber una única persona que adopte todas las decisiones.

**• Estándares**

* Sin estos la reutilización de ideas y componentes no se facilitarían, estos juntan individuos son experiencia, encapsulan buenas ideas y conectan componentes, no obstante, el proceso de crear estándares puede durar mucho, haciendo perder el rumbo a lo que se quiere llegar.

**• Lenguajes específicos del dominio**

* En el entorno del software, ha resurgido el interés por crear lenguajes específicos del dominio (DSL), pequeños lenguajes independientes de creación de secuencias de comandos, que permiten crear código que sea leíble y de manera estructurada, como si lo hubiese hecho un experto.

**EMERGENCIA**

El no darse cuenta de elementos mal ejecutados o establecer duplicados sin verificar o hacer pruebas es lo que conlleva al denominado código sucio.

**• Duplicados**

* Son los mayores enemigos de un sistema bien diseñado. Suponen un esfuerzo adicional, riesgos añadidos y una complejidad a mayores innecesaria, El no eliminarlos generara que haya problemas en un tiempo después.

**• Expresividad**

* Muchos tenemos experiencia con código enrevesado. Muchos lo creamos. Es fácil crear código entendible por nosotros, pero eso no sucede con los demás, lo que para nosotros pueden ser nombres de variables, métodos y clases, que expresen con claridad la funcionalidad del software, para los usuarios u otros allegados que deban dar mantenimiento se les puede resultar complicado examinar y modificar el código, en otras palabras, al momento en el que nos centramos en un código solo legible por nosotros, estamos cayendo en código sucio.

**CONCURRENCIA**

Es la manera de aplicar más métodos y procesos para que el código este más ordenado y agrupado, así sería más fácil corregir errores solo buscando en respectivos procesos. No obstante, esto no necesariamente es aplicado, ya que el hacer más procesos y métodos, esto generara que el creador de ducho código tenga que aplicar rigurosamente los procesos, además de una exhaustiva búsqueda de cualquier error, por lo cual muchos prefieren poner todo y procesarlo en un solo espacio.

**CONCLUSIONES**

* En conclusión, el código sucio es una manifestación que surge cuando la persona que elabora un código en un programa no tiene respeto, orden o el tiempo para dedicarle a la elaboración de dicho código.
* El grupo concluye que el código sucio debe ser evitado a toda costa, puesto que se debe analizar primero el código para recién proseguir a ejecutarlo.
* Finalmente se puede apreciar que este problema llamado código apestoso, genera que una vez diseñado y ejecutado el software, sea cada vez más difícil darle mantenimiento, ya que solo el que lo creo puede entender el código, dándole una desventaja a aquellos usuarios, que no entienden lo que el creador trato de mostrar con su software.

**RECOMENDACIONES**

* El grupo recomienda, ser exhaustivamente minuciosos al momento de analizar y ordenar el código.
* Nuestro equipo recomienda que para mantener alejado un código sucio, este debe ser conciso, con nombres legibles y entendibles, a la vista de cualquier usuario o persona que le de mantenimiento.
* Finalizando, es siempre pensar en los que puedan usar ese programa o software de aquí hacia un futuro, ya q si el apuro por terminar nos hace fallar, terminaremos generando este terrible código.

**REFERENCIAS BIBLIOGRÁFICAS**

* C. MARTIN, Robert. Código Limpio-Manual de estilo para el desarrollo ágil del software. Madrid: Anaya Multimedia, 2012.

ISBN: 9788441532106

* SETA, Leonardo. Las 9 cualidades del Código Limpio [en línea]. [Fecha de consulta: 15 de octubre de 2017].

Disponible en:

<https://dosideas.com/noticias/desarrollo-de-software/965-las-9-cualidades-del-codigo-limpio>

* VILLARES, Raúl. Código sucio [en línea]. [Fecha de consulta: 12 de octubre de 2017].

Disponible en:

<https://medium.com/@raulvillares/c%C3%B3digo-sucio-23bf9cdeaf63>